A Data Stream Processing Optimisation Framework for Edge Computing Applications

Gayashan Amarasinghe, Marcos Dias de Assuncao, Aaron Harwood, Shanika Karunasekera

A Data Stream Processing Optimisation Framework for Edge Computing Applications

Gayashan Amarasinghe*, Marcos D. de Assunção†, Aaron Harwood‡ and Shanika Karunasekera‡
*‡Dept. of Computing and Information Systems, The University of Melbourne, Australia
*gamarasinghe@student.unimelb.edu.au
‡{aharwood, karus}@unimelb.edu.au
†Inria Avalon, LIP Laboratory, ENS Lyon, University of Lyon – France
marcos.dias-de-assuncao@inria.fr

***Abstract*—Data Stream Processing (DSP) is a widely used programming paradigm to process an unbounded event stream. Often, DSP frameworks are deployed on the cloud with a scalable resource model. One of the key requirements of DSP is to produce results with low latency. With the emergence of IoT, many event sources have been located outside the cloud which can result in higher end-to-end latency due to communication overhead. However, due to the abundance of resources at the IoT layer, Edge computing has emerged as a viable computational paradigm. In this paper, we devise an optimisation framework, consisting of a constraint satisfaction formulation and a system model, that aims to minimise end-to-end latency through appropriate placement of DSP operators either on cloud nodes or edge devices, i.e. deployed in an edge-cloud integrated environment. We test our optimisation framework using OMNeT++, with realistic topologies and power consumption data, and show that it is capable of achieving ≈ 1.65 times reduction of latency compared to edge-only and cloud-only placements, which in turn also reduces the energy consumption per event by up to $\approx 4\%$ at the edge layer. To the best of our knowledge our optimisation framework is the first of its kind to integrate power, bandwidth and CPU constraints with latency minimisation.**

I. Introduction

The recent developments on Internet of Things have led to the increasing availability of smart-sensors, wearable devices, smart-home devices, single-board computers, and other various connected smart-apparatus which produce billions of events that need to be analysed by some mechanism. Among the emerging application scenarios that exploit the analysis of such data, IoT can have a disruptive impact on society due to its pervasiveness and ability to connect a vast number of objects to the Internet. For instance, a utility company can monitor the power grid and water distribution network in order to identify possible problems with ageing infrastructure, such as leaks, predict demand and plan accordingly [20].

Several of these emerging application scenarios produce streams of data whose processing is most valuable when carried out under short delays. One of the challenges consists in provisioning the IT resources required to process these data streams. Cloud computing has been used to enable resource sharing, elasticity, data storage, and provide global presence under a pay-as-you-go business model. Although Cloud computing can provide the computing and storage resources required by these applications, emerging services demand data processing and analysis under very short response times. Edge computing [9] offers the ability to deploy services on resources such as micro data centres and IoT gateways that provide non-negligible computing power and are often located closer to where the data is generated (*e.g.* by sensors and small devices). Although edge devices provide lower latency to data sources when compared to the Cloud and their use is often free of charge, offloading data processing to the edges introduces challenges. Many edge devices are connected to the Cloud via wireless networks and are powered by batteries.

We consider the deployment of data stream processing applications using Cloud and edge resources where certain tasks are performed at the edge for reducing the end-to-end latency of processing events. Many Distributed Stream Processing frameworks use a dataflow approach where incoming data traverses a directed graph of operators that execute transformations over the streaming data [1]. During application deployment, such operator tasks are placed onto cluster nodes to benefit from task and data parallelism. The placement of processing tasks across cloud and edge needs to consider computing, network and energy constraints. We model the scenario as a constraint satisfaction problem and employ a solver to acquire placement plans for cloud and edge devices.

More specifically, we make the following contributions: *(i)* we provide an optimisation framework that models the placement scenario of data stream processing applications as a constraint satisfaction problem considering dataflow regions that can be deployed on edge or cloud computing devices, the computing requirements of operators and the power used for computing and communication; and *(ii)* we demonstrate the effectiveness of our framework through the use of the OMNeT++ simulator, where we developed a specific simulation toolkit, which we call ECSSim++, that provides accurate measurements of power consumption and network latency for a given data stream topology allocation and power consumption data. We used a number of realistic data stream topologies, specific to distributed stream processing on edge-cloud environments.

The rest of this paper is organised as follows. Section II describes the optimisation framework, whereas the modelling and simulation tool is described in Section III. The evaluation scenario and performance results are presented in Section IV.

Section V discusses related work and Section VI concludes the paper.

II. Optimisation Framework

Due to the NP-Hard nature of this operator assignment problem [5][6][15], a near optimal task allocation strategy would try to map tasks in the DSP application to the nodes in the host network to improve the quality of service requirements, and reduce the resource consumption. However, the optimality of the task allocation relies on the accuracy of the optimisation framework and the imposed constraints. Therefore, we build an accurate and comprehensive framework that includes a system model and a constraint satisfaction model, to represent the problem using attributes that are critical for distributed stream processing on cloud and edge environments. Before describing the constraints and the optimisation problem, we model the host network, and the DSP application first.

A. System Model

A host network of a distributed stream processing system which consists of both cloud and edge resources, can be represented as a connected graph. Let this graph be $G_{res} = \{V^e_{res}, V^c_{res}, E_{res}\}$ where V^e_{res} represents the set of edge nodes, V^c_{res} represents the set of cloud nodes, and E_{res} represents the set of logical links between the nodes. Let $V = \{V^c_{res}, V^e_{res}\}$ such that $|V| = L$.

A distributed stream processing application, known as a topology, can be represented as a Directed Acyclic Graph (DAG) of source(s), operator(s), and sink(s). Let this graph be $G_{top} = \{So, Op, Si, E_{top}\}$ where So represents the set of sources, Op represents the set of operators, Si represents the set of sinks, and E_{top} represents the set of streaming data flows between the vertices in the graph. So, Op, Si all represent the vertices in the DAG while E_{top} represents the edges of the DAG. Without loss of generality, we also declare the set of vertices in the topology as tasks, T, such that $T = \{So \cup Op \cup Si\}$ and $|T| = K$.

There exists a subset of operators Op^e which are eligible to be edge executable (and optionally cloud executable) where $Op^e \subseteq Op$ and similarly a subset of cloud executable operators Op^c where $Op^c \subseteq Op$ and $Op = Op^c \cup Op^e$.

Next, we define a region, Reg of the distributed stream processing topology as the *closed neighbourhood*[1] of a given vertex (a source, an operator or a sink). For an example, given any operator Op_i in the topology, a region that includes that operator is the closed neighbourhood of that operator, $Reg_{Op_i} = N_{G_{top}}[Op_i]$. With respect to the edge and cloud integrated system, we define an *Edge Executable region*, Reg^e as a region with no cloud executable vertices ($Op^c \cap Reg^e = \emptyset$), and a *Cloud Executable region*, Reg^c as a region with no edge executable vertices ($Op^e \cap Reg^c = \emptyset$) in the topology. Note that both Reg^e and Reg^c can contain sources and/or sinks. Next we define a *source neighbourhood region*, $Reg_{So} = \{Reg_{So_i} | So_i \in So \ \wedge Reg_{So_i} = N_{G_{top}}[So_i]\}$.

[1]A closed neighbourhood of a vertex v in a graph G, denoted by $N_G[v]$, is the induced subgraph of all vertices adjacent to v, including v itself.

We define *Edge Executable, Source Neighbourhood regions (EESN regions)*, Reg^e_{So} as the set of regions with only edge executable operators in closed neighbourhood to a source such that any such region, $Reg^e_{So} = \{Reg^e_{So_i} | \ Reg^e_{So_i} = N_{G_{top}}[So_i] \ \wedge \ Op^c \cap Reg^e_{So_i} = \emptyset\}$. We then define k-EESN regions where $Reg^e_{So_i}(k)$ defines the k-neighbourhood closed graph of node So_i which includes all the nodes in the topology within a diameter k from the source So_i. Therefore the default EESN region defined earlier becomes a special case of k-EESN regions with $k = 1$ such that $Reg^e_{So_i} = Reg^e_{So_i}(1)$.

B. Characteristics of a DSP application

A DSP application consists of four components, source, operator, sink, and the dataflow. In a typical DSP scenario, the sinks, sources and operators, known as tasks, are placed and executed on a processing node. These different tasks in the topology have their own characteristics depending on the application.

A source generates events – therefore has an event generation rate associated with it. The event generation rate vary depending on the DSP application. We can expect a uniform rate if the source is periodically reading a sensor and transmitting that value to the application. If the source is configured to read a sensor and report only if there was a change in reading, it can produce events at a rate that is difficult to predict. If the source is related to human behaviour, e.g.: activity data from a fitness tracker, the event generation rate can follow a bimodal distribution with local maxima, where people are most active in the morning and the evening [19]. Each generated event can have either a fixed size (*e.g.* generated by a temperature sensor) or variable size (*e.g.* a tweet or log collection system) depending on the application.

There are two main operator characteristics that affect the dataflow. For a given operator T_j, the *selectivity ratio* σ_{T_j} is the ratio between the outgoing throughput $\lambda^{out}_{T_j}$ and the ingress throughput $\lambda^{in}_{T_j}$ [16][19], *i.e.* $\sigma_{T_j} = \lambda^{out}_{T_j} / \lambda^{in}_{T_j}$. We also introduce, for a given operator T_j, *productivity ratio* ρ_{T_j}, which is the ratio between an outgoing message size $\beta^{out}_{T_j}$ and the incoming message size $\beta^{in}_{T_j}$, *i.e.* $\rho_{T_j} = \beta^{out}_{T_j} / \beta^{in}_{T_j}$.

The selectivity and productivity ratios depend on the incoming event size and the transformation conducted at the operator [19]. For an example, if the operator is calculating the average of incoming temperature readings for a given time window, an upper bound for the productivity ratio can be estimated since outgoing event structure is known. If the window length is known, the selectivity ratio can also be estimated. However, if the operator is tokenizing a sentence in a textual file, the productivity ratio and the selectivity ratio can vary depending on the input event.

Based on the system model that we have established above, we move on to define the constraints that are relevant to the allocation problem. These constraints are used to limit the search space on potential allocations and to acquire a near optimum allocation to satisfy the requirements.

C. Constraint Satisfaction Problem

In R-Storm[15], Peng *et al.,* formulate a constraint satisfaction problem to model the different resource requirements of the tasks in the topology against the resource availability of a cloud based infrastructure. We use the same approach to model the task placement problem for a heterogeneous network with both cloud and edge nodes comprehensively as a part of our optimisation framework.

A Constraint Satisfaction Problem (CSP) can be defined as $\theta = (\alpha, \beta, \mathbf{C})$ where, $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ (a set of variables), $\beta = \{\beta_1, \beta_2, ..., \beta_n\}$ (a set of respective domains), and $\mathbf{C} = \{\mathbf{C_1}, \mathbf{C_2}, ..., \mathbf{C_k}\}$ (a set of constraints). A solution to a complete and consistent CSP is a task allocation which can be defined as a set of mappings, $A : T \rightarrow V$ such that $T_i \mapsto A(T_i) = \nu_j$. An optimal solution to the CSP problem can be defined as: given an objective function F, where the optimal solution is achieved when F is minimised, find a mapping $A^* \in A$ such that $\forall A, F(A^*) \leq F(A)$.

Furthermore, we say two tasks in a distributed stream processing topology are *collocated* if the two tasks are adjacent in G_{top} and, after the allocation, the tasks are placed on the same node in G_{res}, *i.e.*, vertices $T_i, T_j \in T$ are collocated iff $T_iT_j \in E_{top}$ and $A(T_i) = A(T_j)$.

1) Variables and Domains: To translate the operator placement problem to a constraint satisfaction problem, we introduce K decision variables, $x_i \in X$, where $V^e_{res} = \{V^e_1, V^e_2, \dots V^e_{l_e}\}$, with $|V^e_{res}| = l_e$, and $V^c_{res} = \{V^c_{l_e+1}, V^c_{l_1+2}, \dots V^c_L\}$, with $|V^c_{res}| = L - l_e = l_c$ such that $\forall i \in \{1 \dots K\}$, $x_i = \{1 \dots L\} = \{1 \dots, l_e, l_e + 1, \dots L\}$. We also define $K \times L$ boolean variables $y_{ij} \in Y$ such that $\forall i \in \{1 \dots K\}$ and $\forall j \in \{1 \dots L\}$:

$$y_{ij} = \begin{cases} 1 \ ; \text{if the task } T_i \text{ is allocated to node } V_j \\ 0 \ ; \text{otherwise} \end{cases} \tag{1}$$

Next we define $K \times K$ boolean variables $z_{ij} \in Z$ such that $\forall i, j \in \{1 \dots K\}$ where $T_i, T_j \in T$:

$$z_{ij} = \begin{cases} 1 \ ; \text{if the task } T_i \text{ is adjacent to task } T_j \\ 0 \ ; \text{otherwise} \end{cases} \tag{2}$$

Therefore the variables and domains of the CSP can be translated in to the operator placement problem as $\alpha = \{X, Y, Z\}$ and $\beta = \{\{1 \dots L\}, \{0, 1\}, \{0, 1\}\}$.

Next we look at the relevant constraints of the problem. We introduce two types of constraints here, *Residency constraints* which are the constraints on the location of the operator, and *Resource constraints* which are the constraints on the resource utilisation. We define these constraints on top of our system model.

2) Residency Constraints: First we look at the constraints on the placement of tasks based on the residency requirements. Due to specific hardware, software or other requirements such as privacy or policy requirements, some tasks may be required to be placed, or pinned, on specific resources. Let $\nu \subseteq V_{res}$ be the set of nodes, the tasks T_i are to be pinned on. Then:

$$\forall T_i \in T : A(T_i) \in \nu \tag{3}$$

Since we are considering edge-cloud DSP applications, we can assume that the sources of the topology need to be placed on the edge devices. Therefore we can derive the following constraint from Eq. 3, where $\nu = V^e_{res}$:

$$\forall T_i \in So : T_i : x_i \notin \{l_e + 1 \dots L\}, A(T_i) \in V^e_{res} \tag{4}$$

Similarly the cloud executable tasks in the topology need to be placed only on the cloud nodes. Therefore:

$$\forall T_i \in Reg^c : T_i : x_i \notin \{1 \dots l_e\}, A(T_i) \in V^c_{res} \tag{5}$$

Next we define the residency constraint on collocating the tasks on *k-EESN* regions with the respective source of the region. Here:

$$\forall T_j \in Reg^e_{So_i}(k) \wedge T_i \in So : A(T_j) = A(T_i) \tag{6}$$

Due to reasons similar to pinning tasks on specific resources, some tasks in the DSP topology would be restricted from being placed on certain nodes. We capture this requirement here. Let $\overline{\nu} \subseteq V_{res}$ be the set of restricted nodes per each task $T_i \in T$, then:

$$\forall T_i \in T : A(T_i) \in V_{res} - \overline{\nu} \tag{7}$$

3) Resource Constraints: When placing tasks on a cloud and edge integrated environment, the resource requirements become a critical component in the decision making process. We look at several resources in this regard, CPU, bandwidth, and energy.

Let $\zeta_{T_iV_j}$ be the number of CPU cycles needed to process an incoming event on task T_i for the specific hardware architecture of node V_j, and $\lambda^{in}_{T_i}$ be the ingress throughput (events/s) of task T_i. Let C_{V_j} be the clock speed of the CPU in Hz, $\eta_{V_j} \geq 1$ be the number of cores on node V_j, ψ_{V_j} be the number of threads executed per core, and $w_C > 0$ denote the hardness of the CPU constraint. Then:

$$\forall V_j \in V_{res} : \ \frac{1}{\eta_{V_j}\psi_{V_j}} \sum_{\forall T_i} y_{ij}\zeta_{T_iV_j}\lambda^{in}_{T_i} \leq w_C C_{V_j} \tag{8}$$

Let $\beta^{out}_{T_i}$ be the size of an outgoing event (kB/event) at task T_i and $\lambda^{out}_{T_i}$ be the outgoing throughput (events/s) of task T_i. Let $B^{out}_{V_jV_k}$ be the bandwidth on edge $V_jV_k \in E_{res}$, and $w_B > 0$ be the hardness of bandwidth constraint. Then:

$$\begin{aligned} &\forall V_j, V_k \in V_{res}, \forall T_i \in T : \\ &\quad y_{ij}\beta^{out}_{T_i}\lambda^{out}_{T_i} z_{i(i+1)} y_{(i+1)k} \leq w_B B^{out}_{V_jV_k} \end{aligned} \tag{9}$$

The energy consumption for an incoming event can be divided into three sections: (i) the energy consumed while receiving the event, (ii) the energy required to process the event, and (iii) the energy required to transmit the result to the next node. Let $\mu^r_{V_{k-1}V_j}$ (J/event) be the energy consumed when receiving an event at node V_j from node V_{k-1}, $\mu^p_{T_iV_j}$ (J/event) be the energy consumed for processing an event on task T_i on the node V_j, and $\mu^t_{V_jV_k}$ (J/event) be the energy consumed for transmitting an event from node V_j to node V_k. Let P_{V_j} (W) be the maximum power draw from the power source at

node V_j, and let $w_P > 0$ be the hardness of power constraint. Then:

$$\forall V_j \in V_{res} : \sum_{\forall V_{k-1}} \sum_{\forall T_{i-1}} y_{(i-1)(k-1)} z_{(i-1)i} y_{ij} \mu^r_{V_{k-1}V_j} \lambda^{in}_{T_i} + \sum_{\forall V_k} \sum_{\forall T_i} y_{ij} z_{i(i+1)} y_{(i+1)k} \mu^t_{V_jV_k} \lambda^{out}_{T_i} + \sum_{\forall T_i} y_{ij} \mu^p_{T_iV_j} \lambda^{in}_{T_i} \leq w_P P_{V_j} \tag{10}$$

D. Optimisation Problem

The aim of our optimisation framework is to minimise the end-to-end latency of the DSP topology as it runs on the underlying network, since edge-cloud applications are known to suffer from high end-to-end latency [18]. Loosely, the end-to-end latency is the time between an input tuple being generated at a source node to when the results concerning that input tuple arrive at the sink node. We have found however that it is problematic to formulate this as an expression. We considered minimising the latency of the critical path in the topology, but doing so does not take into account the interactions between paths at operators, and such a minimisation function failed to achieve our aim. For an example an aggregation operator with multiple incoming edges may wait for the events from all the edges to produce a resulting event. Therefore as a heuristic function, we considered *minimising the sum of individual latencies* over the topology and indeed in our experiments we have found that this objective function consistently gives the lowest end-to-end latency. We breakdown the latency calculation into two parts, the computational latency and network transfer latency, and thereby calculate the sum of latencies:

$$\mathbf{L} = \sum_{T_i \in T, V_j \in V_{res}} y_{ij} \delta_{T_iV_j} + \sum_{V_j, V_k \in V_{res}} \sum_{T_i, T_{i+1} \in T} y_{ij} z_{i(i+1)} y_{(i+1)k} \vartheta_{V_jV_k} \tag{11}$$

where $\delta_{T_iV_j}$ is the processing time of task T_i at the host V_j, and $\vartheta_{V_jV_k}$ is the latency of transferring an event between two hosts. Then our optimisation objective is to minimise $\mathbf{L}$ of G_{top} placed on G_{res}.

III. A Toolkit for Modelling and Simulation of Data Stream Processing

ECSSim++ is a toolkit that we designed and implemented to simulate the execution of a DSP application on distributed cloud and edge environments. The toolkit was implemented on top of the OMNeT++ framework[2][22] employing the native network simulation capabilities of the INET framework[3]. OMNeT++ enabled the development of a modular, component based toolkit to simulate the inherent characteristics of edge-cloud streaming applications and networks. The following subsections detail how the different elements of a stream processing solution are modelled.

[2]https://omnetpp.org/

[3]https://inet.omnetpp.org/

A. Host Network and Processing Nodes

We implemented two types of processing nodes, namely cloud nodes and edge nodes. As shown in Fig.1a, the edge computing nodes are connected to the network via IEEE 802.11 WLANs using access points for each separate network. Depending on the scalability of the edge layer, the number of access points can be configured. All the wired connections, used mostly by the cloud nodes, are simulated as full duplex Ethernet links. All the communications in the host network are simulated using the UDP protocol. In addition to the lower layer capabilities, each device is configured with a UDP application layer to allow the DSP application components to communicate over the host network.

B. DSP Application

Three separate task modules were developed to represent sources, operators, and sinks. A task module can be placed on a processing node and can be connected according to the adjacency matrix of the DSP topology, to simulate the dataflow of a DSP application. Instances of these modules should be configured, depending on the characteristics of each task in the DSP application topology. For each source, the event generation rate and the initial event size has to be configured. Similarly, the selectivity and the productivity ratios of each operator in the topology must be specified.

C. Processing Model

Each cloud/edge node has a processing model to simulate a multi-core, multi-threaded CPU. A node is configured with 3 parameters; η_{V_j}, C_{V_j}, and ψ_{V_j}. A CPU core module simulates a single core instance.

When an event arrives at an operator for processing, the operator selects a CPU core depending on the scheduling policy. We implemented a *round-robin scheduler* for this study, but it is possible to extend this behaviour and implement other policies. Once a CPU core is selected, the operator assigns the event to the core module, which places it in a single FIFO queue for each operator simulating the behaviour of a multi-threaded execution where the execution of each operator is done in a single thread. To simulate the processing of an event at a particular operator, we utilise a property assigned by each operator for an incoming event – the number of CPU cycles required to process the particular event in the given CPU, $\zeta_{T_iV_j}$. This value depends on the processor architecture and can be determined empirically by profiling an operator [4] or even by simply analysing the compiled assembly code, depending on the complexity of the application code. The event to be processed, will be held in the queue inside the CPU core for a time period $\delta_{T_iV_j}$ which can be calculated with the following equation. After holding the event in the queue for $\delta_{T_iV_j}$, the CPU core pops the event from the queue, and sends it back to the operator that scheduled it, which completes the processing of the particular event on that operator.

$$\delta_{T_iV_j} = \frac{\zeta_{T_iV_j} \times \text{No of threads running on the CPU core}}{C_{V_j} \times \psi_{V_j}} \tag{12}$$

When an operator receives a processed event, it sends the event to the next operator or sink downstream. And each event is processed in this manner until it reaches a sink.

D. Power Model

As we are mainly interested in the power consumption of edge devices, we have limited the power consumption model to represent the power consumption of wireless communication between the edge devices and the access points, and the power consumed by their CPUs while processing data events. However, the power model can be extended to other processing requirements and communication media as well.

The power consumed by an edge device during communication is computed by a component responsible for measuring the radio state changes. A network card is considered to have an idle or static consumption, and a dynamic consumption when transmitting or receiving data. These parameters are configurable and can be measured or estimated based on empirical data [10][11].

To model the power consumed by the CPU, we consider that a CPU core operates within two states, an idle state and a busy state. When the CPU core is processing an incoming event, as per section III-C, it operates in the busy state, and otherwise it is idle. The amount of power consumed when busy, depends on the utilisation which can be measured or estimated [10], [11]. The total power, Φ_{total} consumed at each edge device can be calculated by, $\Phi_{total} = \Phi_{idle} + \Phi_{cpu}(u) + \Phi_{wifi,idle} + \Phi_{wifi,up} + \Phi_{wifi,dn}$ where u is the CPU utilisation of the device.

IV. Evaluation

Our experiments are based on using the placement plans generated by the optimisation framework introduced in Section II. The plans are used as the input for the simulator introduced in Section III. We measure the **end-to-end delay** of processing an event and the **energy consumption** of the edge devices during the execution. The presented results are averages of 5 experimental runs varying the simulation seed with simulation configurations of Table II.

A. Placement Strategies

We employ two categories of DSP application topologies. The first category is a set of synthetic micro-benchmark topologies: linear, diamond, and tree. A similar micro-benchmark based analysis approach is proposed by Peng *et al.,* [15]. However, we have implemented our micro-benchmarks to represent common DSP application topologies with characteristics specific to edge-cloud integrated stream processing by adopting some of the topologies from other work [6][17][19]. We use these micro-benchmarks to establish a baseline for our optimisation framework. As the second category we use a realistic edge-cloud application topology. We use the ETL+STATS topology from the RIoTBench suite [19] for this purpose.

Here we assume the event generation rate is uniform and the initial event size is fixed and known. The set of topologies used for this simulation follow the same characteristics. We also assume the productivity ratio and selectivity ratio for each operator is fixed and known for the considered DSP topologies.

For each DSP topology, we execute 3 placement plans separately to test our hypothesis; *Cloud-Only (CO)* plan places all source nodes of the topology on the edge layer while the rest of the processing happens at the cloud layer, *Edge-Only (EO)* plan places everything except the sink nodes in the topology on the edge layer, the sink nodes are placed on the cloud layer, *Framework-Optimised (FO)* plan places the placement allocation generated by our optimisation framework on the topology, which includes a shared placement of nodes in the topology between the edge and the cloud layer, e.g. as shown in Fig.1b and Fig.1c.

To obtain the FO placement plans for each topology, we implemented our optimisation framework using the MiniZinc constraint modelling language [14][21]. Then we solve the model using the open-source Gecode solver to acquire each FO placement plan.

B. Simulation Setup

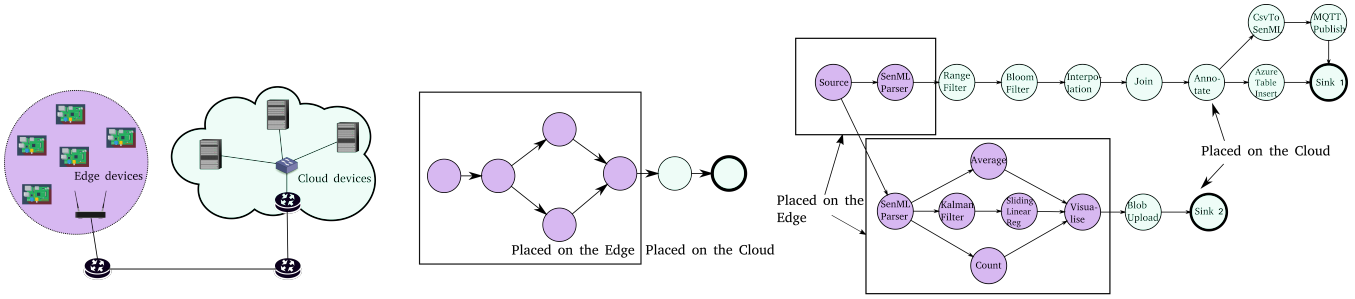
We use the characteristics of Raspberry Pi Model B (PiB) devices and Raspberry Pi 3 Model B (Pi3B) devices as our edge nodes, and we use the configurations of Intel Xeon Gold 6140 processor to model the processing capability of our cloud nodes in ECSSim++ toolkit. We run 2 sets of separate experiments with PiB and Pi3B devices as edge nodes (see Table I).

When configuring the power model of the ECSSim++ simulator, we use the power models generated in the work of Kaup *et al.,* [10][11] as the power model of our Raspberry Pi devices as shown in Table III.

C. Results

1) Performance of the Micro-benchmarks: We first evaluate the effectiveness of the FO placement generated by our optimisation framework for the 3 synthetic micro-benchmark topologies in our simulation.

Fig.2 illustrates the end-to-end latency improvement of the micro-benchmark topologies. As observed here, for all 3 micro-benchmark topologies on both PiB and Pi3B devices, the Framework-Optimised (FO) placement results in $\approx 1.8-2$ times reduction when compared with the Cloud-Only (CO) placement. When you compare the FO placement against the naive Edge-Only (EO) placement, in the Linear topology the reduction is ≈ 1.2 times. In the Tree topology the reduction is ≈ 1.08 times, while in the Diamond topology the reduction is slightly more than ≈ 1.01 times. These results show that offloading some tasks to the edge devices yields lower end-to-end latency depending on the application. However, a naive placement decision of placing all the tasks except the sinks on the edge devices does not guarantee better end-to-end latency since there are other factors that affect the placement decision, such as the computation, bandwidth and power consumption requirements of the tasks, as well as other constraints that affect their placement. Therefore, our experiments demonstrate



(a) Host network with edge and cloud resources

(b) Framework-Optimised placement of a synthetic micro-benchmark topology

(c) Framework-Optimised placement of ETL+STATS application topology

Fig. 1: Host network and the task placement

TABLE I: Configurations for Host Nodes

	PiB	Pi3B	Cloud Node
Released Year	2012	2016	2017
Processor Freq.	700MHz	1.2GHz	2.3GHz
CPU Cores	1	4	18
RAM	512MB	1GB	16GB
Threads per core	1	1	1

TABLE II: Simulation Parameters

Parameter	Value
# Events	100000
# Edge Devices	50
# Cloud Devices	5
Warm-up Period	2s
WAN Bandwidth	6Mbps
LAN Bandwidth	100Gbps

TABLE III: Power consumption configuration of Raspberry Pi devices

Parameter	Power Consumption(mW)	
	PiB	Pi3B
Φ_{idle}	1577	1488
$\Phi_{wifi,idle}$	942	764.5
$\Phi_{wifi,dn}$	1020.61	822.1
$\Phi_{wifi,up}$	1112.38	704.431
$\Phi_{cpu}(u = 95\%)$	171.95	588.145
$\Phi_{cpu}(u = 5\%)$	9.05	30.955

that the FO placement plans generated by the optimisation framework, results in achieving better end-to-end latency results in these micro-benchmark topologies, where optimising the sum of individual latencies produce lower end-to-end latencies.

In the Fig.2, we also observe that the Pi3B devices produce lower latencies than the PiB devices. The reason for this observation is that the Pi3B devices have significantly more processing power when compared with PiB devices (1.2 GHz 4 cores vs 700 Mhz 1 core). Hence the overall processing latency is lesser in the Pi3B devices which in turn results in producing less overall mean end-to-end latency than the PiB devices. We keep all the other parameters such as the network topology, network bandwidth, and the cloud nodes the same in both edge device scenarios.

Table IV shows the per-event average energy consumption of the 3 micro-benchmarks on PiB and Pi3B devices with each placement plan. The total average per-event energy consumption is the sum of the processing energy consumption and the network energy consumption. Overall, the processing energy consumption is highest in the EO placement which is followed by the FO placement and then the CO placement. In the EO placement, all the tasks except the sinks are placed on the edge, hence the highest amount of processing is done at the edge which results in the higher processing energy consumption. And similarly the FO placement uses the next highest edge processing energy where the CO placement uses the lowest amount of processing since only the sources are located on the edge.

However, the network energy consumption cannot be compared trivially since the number of events emitted by the last task on the edge device and the size of the emitted events dictate the amount of energy consumed for transmitting these events, in addition to the other network communication overheads.

When the total energy consumption of the 3 micro-benchmark topologies are compared we can observe that the FO placement results in lower energy consumption than the CO placement. However, when compared with the naive EO placement, the FO placement performs better in all scenarios except for the Diamond and Tree topology on the PiB devices, and the Diamond topology on the Pi3B devices. Even though the processing energy consumption is less in the FO placement here, if the last task on the edge device is transmitting more events to the cloud, the network energy consumption is higher on these devices. It is important to note that this does not violate our optimisation goal since still the end-to-end latency is lowest in the FO placement while the power constraint is satisfied.

2) Performance of the ETL+STATS application: We use the ETL+STATS application topology from the RIoTBench suite [19] as our application scenario. In order to simulate the event stream, we use the characteristics of the FIT dataset with a peak event rate of 500 events/s and source event size of 1024 bytes, from [2][19]. As shown in Fig.1c, ETL+STATS application topology consists of 19 tasks in total, with 1 source, 16 operators and 2 sinks.

Fig.3 illustrates that the FO placement plan results in a mean end-to-end latency reduction of $\approx 1.54 - 1.57$ times against the CO placement plan and $\approx 1.68 - 1.72$ times against the EO placement plan. It is important to note that the naive EO placement plan generates worst latency performance in this application, mainly due to the complexity of the topology, where a naive placement is not viable due to many parameters

TABLE IV: Per-event average energy consumption of micro-benchmark topologies and the application topology (Values in bold show the minimum energy consumption of each topology on each edge device.)

Edge Device	Topology	Plan	Energy Consumption (mJ/event)		
			Processing	Network	Total
PiB	Linear	CO	4.7572	18.0047	22.7619
		EO	4.7577	17.3956	22.1533
		FO	4.7574	16.6788	**21.4362**
	Diamond	CO	4.7572	18.0047	22.7619
		EO	4.7577	16.715	**21.4727**
		FO	4.7575	17.0586	21.8161
	Tree	CO	2.3787	17.3625	19.7412
		EO	2.3790	16.3860	**18.7650**
		FO	2.3789	16.4862	18.8651
	Application	CO	4.7573	18.3592	23.1165
		EO	4.7581	18.2682	23.0263
		FO	4.7577	17.396	**22.1537**
Pi3B	Linear	CO	4.5560	16.8250	21.3810
		EO	4.5569	16.2078	20.7647
		FO	4.5563	15.7779	**20.3342**
	Diamond	CO	4.5560	16.8250	21.3810
		EO	4.5569	15.4935	**20.0504**
		FO	4.5565	15.6222	20.1787
	Tree	CO	2.2781	16.2956	18.5737
		EO	2.2787	15.5276	17.8063
		FO	2.2785	15.3055	**17.5840**
	Application	CO	4.5561	16.8199	21.376
		EO	4.5576	16.6671	21.2247
		FO	4.5566	16.5012	**21.0578**

that affect the decision. This shows that in a complex topology, the optimisation of sum of individual latencies results in producing better end-to-end latencies.

As seen in Table IV, $\approx 1.5 - 4\%$ improvement can be achieved in per-event average energy consumption of the FO plan agains the CO and EO plans. Since in stream processing an unbounded event stream is processed, even a slight per-event energy consumption reduction results in substantial energy savings on the edge devices. Therefore with this simulated empirical evidence, we show that our optimisation framework is able to produce a placement plan which improves both the end-to-end latency and the per-event edge energy consumption.

V. Related Work

This section discusses *(i)* operator placement strategies, *(ii)* distributed data stream processing in heterogeneous environments (*e.g.* comprising edge and cloud resources).

1) Operator Placement Strategies: Operator placement has been a highly discussed problem with several solutions based on different application scenarios, frameworks, topology structures, underlying infrastructure and expected outcomes [12]. Multiple techniques for operator placement have been proposed in the literature [7], [24].

R-Storm [15] provides a resource-aware scheduler for Apache Storm based on task requirements and the availability of three resource types – CPU, memory, and bandwidth. It proposes a formulation that corresponds to a Quadratic Multiple 3-Dimensional Knapsack problem and offers an approximation algorithm for calculating the operator placement by finding the distance between the resource demands and resource availability in a 3D-resource space. Similar to R-Storm, we model the placement scenario as a constraint satisfaction problem, but our model focuses on environments comprising edge and cloud resources.

Ghosh *et al.,* [8] present a placement strategy for complex event processing that considers computing and network latencies, throughput, and power usage at each cluster node. The authors use genetic algorithms (GA) to acquire a fast approximation to the problem and compare it against a brute force (BF) method that becomes intractable with large problem sizes. Their analysis focuses on the efficiency of the GA approach. Cardellini *et al.,* [5][6] focus on modelling the operator placement problem by considering the availability of nodes, cost of operator execution on nodes, response time, and network QoS metrics. Their work does not focus on applications sharing both edge and cloud resources. Rychly *et al.,*[17] handle operator scheduling on heterogeneous resources. Focusing on incorporating Graphic Processing Units (GPUs), they consider that scheduling decisions are aware of the performance characteristics of individual cluster nodes, of the incoming data, and the topology.

To the best of our knowledge, these studies do not consider the heterogeneity introduced by an edge-cloud environment, the challenges they impose on DSP applications, and have not conducted detailed experiments on edge-cloud environments. We comprehensively model the edge-cloud environment and DSP applications using a constraint based model and evaluate the performance on an edge and cloud integrated network using appropriate micro-benchmarks and real world applications. We analyse the impact these constraints have on placement decisions in terms of end-to-end latency of processing events and energy consumption.

2) Distributed Stream Processing on the Edge: RIoT-Bench[19] is a set of benchmark DSP topologies for real-time IoT applications. Mobistreams [23] focuses on using mobile devices for stream processing to reduce the overhead on the cellular network. The mobile network is divided into regions to which parts of a dataflow are assigned. The work focuses on checkpointing and reliability mechanisms. Using mobile devices for stream processing, Morales *et al.,*[13] also focus on improving fault tolerance mechanisms.

In addition, Beck *et al.,* propose a taxonomy of *Mobile Edge Computing (MEC)* [3]. They classify potential mobile edge computing applications based on 4 metrics of feasibility: power consumption, latency, bandwidth utilization, and scalability. They also identify that the main advantages gained through mobile edge computing are low latency results, offload-ability of workloads, and distribution of processing tasks. Even though the authors do not specifically consider any streaming applications, we believe the same conclusions can be applied in stream processing applications as well. However, the authors limit the concept of membership of *Mobile Edge Computing* only to the base stations of cellular networks. In our work, any device with the capability to execute an application, and logically situated below the cloud, can be a

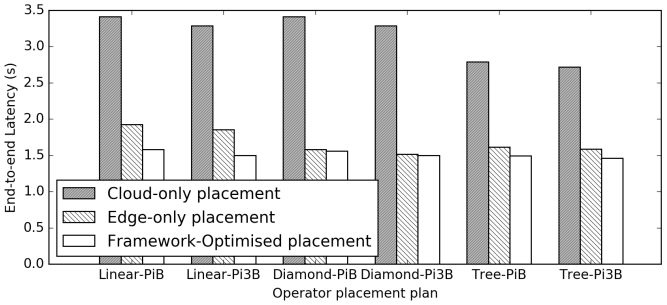


Fig. 2: Mean end-to-end latency of micro-benchmarks

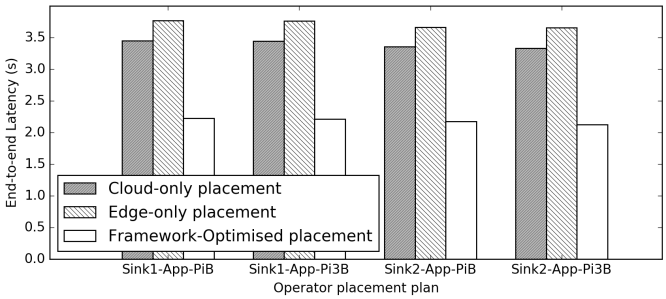


Fig. 3: Mean end-to-end latency of ETL+STATS topo.

member of the edge.

VI. Conclusion

We have introduced a comprehensive optimisation framework that consists of a system model and a constraint satisfaction formulation, to represent the characteristics of DSP applications executed on edge-cloud environments. We consider the characteristics of the host network and the DSP application topology in our system model. Based on this system model, we introduce a set of resource constraints and residency constraints to represent the requirements of executing an application on an edge-cloud environment. By using our optimisation framework to optimise for the sum of individual latencies, and the ECSSim++ simulation toolkit to simulate our edge-cloud DSP application scenario with a realistic host network, we show that we can reduce end-to-end latencies of DSP applications by offloading some tasks to the edge nodes which in turn reduces the per-event energy consumption at the edge.

We show that the generated Framework-Optimised placement plans for the simple micro-benchmark topologies provide slight improvements against a naive Edge-Only placement. However, in a complex realistic application topology, the FO plan results in ≈ 1.65 times reduction of mean end-to-end latency, and $\approx 1.5 - 4\%$ improvement in terms of total per-event energy consumption in the edge devices, which is substantial in a DSP application that processes an unbounded event stream.

Our study opens up many future research directions which include executing our solution in a real edge-cloud deployment to further develop our experiments on real application scenarios. We would like to investigate further the implications of optimising the sum of individual latencies with regard to optimising the end-to-end latencies. In addition, we would like to consider edge-cloud applications involving non-uniform event generation distributions, and event size distributions. We will also study the effects of different optimisation goals on our optimisation framework.

References

[1] M. D. de Assunção *et al.*, "Distributed data stream processing and edge computing: A survey on resource elasticity and future directions," *Journal of Network and Computer Applications*, vol. 103, pp. 1–17, 2018.

[2] O. Banos *et al.*, "Mhealthdroid : a novel framework for agile development of mobile health applications," in *IWAAL*, 2014, pp. 91–98.

[3] M. T. Beck *et al.*, "Mobile edge computing : a taxonomy," *AFIN*, no. c, pp. 48–54, 2014.

[4] A. Camesi *et al.*, "Continuous bytecode instruction counting for cpu consumption estimation," in *Third International Conference on the Quantitative Evaluation of Systems, QEST*, 2006, pp. 19–28.

[5] V. Cardellini *et al.*, "Optimal operator placement for distributed stream processing applications," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, 2016, pp. 69–80.

[6] ——, "Optimal operator replication and placement for distributed stream processing systems," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, 2017, pp. 11–22.

[7] R. Eidenbenz and T. Locher, "Task allocation for distributed stream processing," in *IEEE INFOCOM 2016*, 2016, pp. 1–9. arXiv: 1601.06060.

[8] R. Ghosh and Y. Simmhan, "Distributed scheduling of event analytics across edge and cloud," *CoRR*, pp. 1–31, 2016. arXiv: 1608.01537.

[9] Y. C. Hu *et al.*, "Mobile edge computing: A key technology towards 5G," European Telecommunications Standards Institute (ETSI), Whitepaper ETSI White Paper No. 11, 2015.

[10] F. Kaup, "Energy-efficiency and performance in communication networks: Analyzing energy-performance trade-offs in communication networks and their implications on future network structure and management," PhD thesis, Technische Universität, Darmstadt, 2017.

[11] F. Kaup *et al.*, "Powerpi: measuring and modeling the power consumption of the raspberry pi," in *39th Annual IEEE LCN*, 2014, pp. 236–243.

[12] G. T. Lakshmanan *et al.*, "Placement strategies for internet-scale data stream systems," *IEEE Internet Computing*, vol. 12, no. 6, pp. 50–60, 2008.

[13] J. Morales *et al.*, "Symbiosis: sharing mobile resources for stream processing," in *ISCC*, vol. Workshops, 2014.

[14] N. Nethercote *et al.*, "Minizinc: Towards a standard cp modelling language," *CP 2007*, pp. 529–543, 2007.

[15] B. Peng *et al.*, "R-storm: resource-aware scheduling in storm," in *Middleware '15*, 2015, pp. 149–161.

[16] P. Pietzuch *et al.*, "Network-aware operator placement for stream-processing systems," in *ICDE*, vol. 2006, 2006, p. 49. arXiv: 9780201398298.

[17] M. Rychly *et al.*, "Scheduling decisions in stream processing on heterogeneous clusters," in *CISIS*, 2014, pp. 614–619.

[18] W. Shi *et al.*, "Edge computing: vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[19] A. Shukla *et al.*, "Riotbench: a real-time iot benchmark for distributed stream processing platforms," 2017. arXiv: 1701.08530.

[20] Y. Simmhan *et al.*, "Cloud-based software platform for big data analytics in smart grids," *Computing in Science Engineering*, vol. 15, no. 4, pp. 38–47, 2013.

[21] P. J. Stuckey *et al.*, "The minizinc challenge 2008–2013," *AI Magazine*, vol. 35, no. 2, pp. 55–60, 2014.

[22] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the First International ICST Conference on Simulation Tools and Techniques for Communications Networks and Systems*, 2008.

[23] H. Wang and L. S. Peh, "Mobistreams: a reliable distributed stream processing system for mobile devices," in *IPDPS*, 2014, pp. 51–60.

[24] Y. Xing *et al.*, "Dynamic load distribution in the borealis stream processor," in *ICDE*, 2005, pp. 791–802.